DESIGN CHECKLISTS

1.0 GENERAL

- ☐ Does the design trace to the requirements?
- ☐ Have any unnecessary requirements been added?
- ☐ Does the design satisfy the requirements?
- ☐ Are all software components independent?
- ☐ Have all external interfaces been defined?
- ☐ Is the data structure consistent with the information domain?
- ☐ Do data structures support the logical data architecture access and distribution requirements?
- ☐ Is the design modular?
- ☐ Do software components support the logical architecture user access needs?
- ☐ Is the logical complexity reasonable?
- ☐ Are all algorithms logic correct and produce the desired effect?
- ☐ Are all error and boundary conditions defined?
- ☐ Is compound logic minimized?
- ☐ Is the design amenable to the implementation language?
- ☐ Have language dependencies been minimized?
- ☐ Have all external interfaces been defined?
- ☐ Have all internal interfaces been defined?
- ☐ Have any new PVCS items been identified?
- ☐ Are the system components functionally independent?
- ☐ Is the overall design factored (i.e. top level modules decide program flow; bottom level modules perform I/O and computational work)?
- ☐ Has reuse of existing materials been considered?
- ☐ Has component usability been considered?
- ☐ Has memory utilization been estimated and found acceptable?
- ☐ Has performance been estimated and found acceptable?
- ☐ Is the user interface usable and consistent throughout the software?
- ☐ Has software maintainability been considered?
- ☐ Has any prototyping been performed?
- ☐ If incremental development is planned, is the “build plan” reasonable?

2.0 CLASSES

- ☐ Are the class diagrams and the class specifications consistent with each other (class names, member functions)?
- ☐ Are the relationships between classes clear from the class diagrams?
- ☐ re the classes complete? Consider both attributes (data members) and interfaces.
- ☐ Do class names, method names, and attribute names (data members) conform to the established standard?

Does each class specification include:

- ☐ a clear description of the class purpose?
- ☐ a general description of all necessary class data members (attributes)?
- ☐ name and description of public class member functions?
- ☐ a list of messages generated by the class (methods [class and function name] called in other classes by this class)?
- ☐ Are the classes appropriately independent of each other? [This does not preclude composition (has-a) and inheritance (is-a).]
- ☐ Does the design exhibit a proper level of Modularity?
- ☐ Is each class's purpose clear and complete? Correct? Concise?
- ☐ Does the program do one thing well, not too many things? Is the program coherent?
- ☐ Does each class do one thing well, not too many things? Is each class coherent?
- ☐ Is a class doing anything it shouldn't?
- ☐ Are the new classes re-usable?
- ☐ Do classes provide Encapsulation of data and methods?
- ☐ Do classes exhibit proper Information Hiding?

3.0 INTERFACE AND HIGH-LEVEL DESIGN REVIEWS

Public

- ☐ Is the design fully implementable?
- ☐ Is the design modular?
- ☐ Has re-use of appropriate existing materials been considered?
- ☐ Has system maintainability been considered?
- ☐ Have all external interfaces been defined?
- ☐ Are the system components functionally independent?
- ☐ Is the overall design factored (i.e. top level modules decide program flow; bottom level modules perform I/O and computational work)?
- ☐ Has component [used to say "system"] usability been considered?
- ☐ Have all internal interfaces been defined?
- ☐ Do data structures support the logical data architecture access and distribution requirement?
- ☐ Do system components support the logical process architecture user access needs?
- ☐ Does the implementation architecture support the organizational, geographic, processing, data access, communication, and support requirements of the proposed design?
- ☐ Does the interface design trace to the high-level design?
- ☐ Is the interface design consistent with the high-level design?

Private

- ☐ Does the design trace to the requirements specification?
- ☐ Have any unnecessary requirements been added?
- ☐ Does the design satisfy all allocated software requirements?
- ☐ Has performance been estimated and found acceptable?
- ☐ Has memory utilization been estimated and found acceptable?
- ☐ Have all new techniques been successfully prototyped?
- ☐ Are the ABC charts and the class specs consistent with each other?
- ☐ Are the classes complete? Consider both attributes (data members) and interfaces.
- ☐ Are the classes appropriately independent of each other? This does not preclude composition (has-a) and inheritance (is-a). Modularity.
- ☐ Is each class's purpose clear and complete? Correct? Concise?
- ☐ Does the program do one thing well, not too many things? Is the

program coherent?

- ☐ Does each class do one thing well, not too many things? Is each class coherent?
- ☐ Is the class doing anything it shouldn't?
- ☐ Are we missing any re-use opportunity? Are we re-inventing the wheel?
- ☐ Are the new classes re-usable?
- ☐ Encapsulation.
- ☐ Information hiding.

4.0 INTERFACE DESIGN REVIEWS

Public

- ☐ Is the design fully implementable?
- ☐ Is the design modular?
- ☐ Has re-use of appropriate existing materials been considered?
- ☐ Has system maintainability been considered?
- ☐ Have all external interfaces been defined?
- ☐ Are the system components functionally independent?
- ☐ Is the overall design factored (i.e. top level modules decide program flow; bottom level modules perform I/O and computational work)?
- ☐ Has component [used to say "system"] usability been considered?
- ☐ Have all internal interfaces been defined?

Private

- ☐ Has performance been estimated and found acceptable?
- ☐ Has memory utilization been estimated and found acceptable?
- ☐ 6. H14. Have all new techniques been successfully prototyped?

5.0 DETAILED DESIGN REVIEWS

Public

greater emphasis

- ☐ Is the design fully implementable?
- ☐ Has information hiding been fully utilized?
- ☐ Is the logical complexity reasonable?
- ☐ Have [language/]operating system dependencies been minimized?
- ☐ Are algorithms logically correct; do they produce the desired effect?
- ☐ Have all error and boundary conditions been satisfied?
  - appropriate error reporting techniques (return code vs.exception).
  - error handling (catch and handle, catch and re-throw, catch and throw new, let pass).
  - error logging.
- ☐ Do candidate software materials [what we decide are utilities] adhere to software standards?
- ☐ Is each candidate software component [what we decide are utilities] well documented?
- ☐ Are candidate re-usable software components [what we decide are utilities] structured to be maintainable?
- ☐ Is the implementation of the candiadte software components [what we decide are utilities] efficient?
- ☐ Do the candidate re-usable software components perform their advertized functions correctly and completely? [could apply to both to our deciding what are utilities as well as what COTS and freeware we should use]

- ☐ check for missing/extra #includes.
- ☐ check for include guards.
- ☐ check for missing/unused member data.
- ☐ check for missing local declares (scope issues).
- ☐ check for signed/unsigned integer types.
- ☐ are header file prologues understandable?
- ☐ STL iterators can never be zero (we can't see the internal representation); we can neither assign zero to them nor test them against zero.
- ☐ delete does not zero the pointer; check for de-referencing deleted pointers.
- ☐ check for redundant pdl; factor such pdl out.
- ☐ check for references on const scalar args (e.g. const int &arg1); they are wasteful.
- ☐ check for returning pointers or references to local variables (bad!).

- ☐ check for "overkill" (such as using an STL array when a small simple array will readily do).
- ☐ check for const correctness.
- ☐ check that ints can't be > 32767 (signed) or 65535 (unsigned); use longs where violations are possible.
- ☐ no shorts (use ints).

lesser emphasis

- ☐ Are the interfaces consistent with the interface interface design?
- ☐ Is the design amenable to the implementation language?
- ☐ Are structured programming constructs used throughout?
- ☐ Is compound logic [if(f(x))] minimized?
- ☐ Has inverse logic [nots] been eliminated, or at least minimized?
- ☐ Are local data structures properly defined?
- ☐ Has maintainability been considered?
- ☐ Have all reasonable sources of re-usable components [COTS and freeware] been explored?

Private

- ☐ Does the detailed design trace to the interface and high-level designs?
- ☐ Does the design satisfy all allocated software requirements?
- ☐ Has performance been estimated and found acceptable?
- ☐ Have all categories of potentially re-usable materials been identified? [could apply to both to our deciding what are utilities as well as what COTS and freeware we should use]
- ☐ Have criteria been established for the selection of candidate re-usable materials? [could apply to both to our deciding what are utilities as well as what COTS and freeware we should use]